HoGent

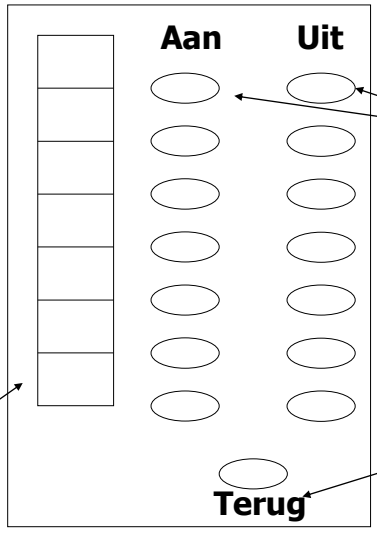
BEDRIJF
EN
ORGANISATIE

Command Pattern

Aanroepen afschermen
Een afstandsbediening voor Home Automation or Bust, Inc.

Voorbeeld “afstandsbediening”

We beschikken over zeven programmeerbare slots. We kunnen ieder slot met een ander apparaat verbinden en besturen via de knoppen.



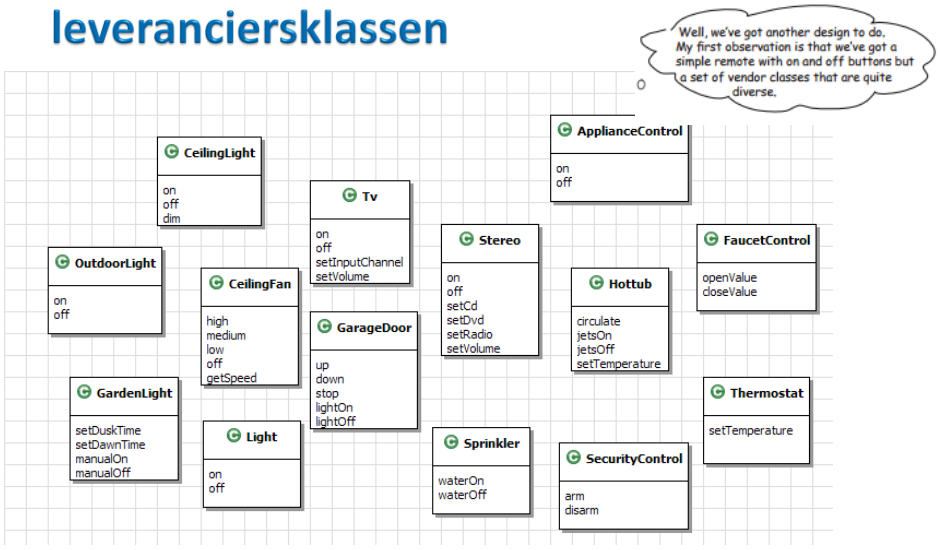
Er zijn ‘aan’ en ‘uit’ knoppen voor ieder van de zeven slots

Deze twee knoppen worden gebruikt voor de besturing van het huishoudelijke apparaat in slot één …
etc.

Hier kun je met viltstift de namen van de apparaten noteren.

Dit is de globale ‘undo’-knop. Deze maakt de functie van effecten van de laatst ingedrukte knop ongedaan.

Home automation : een blik op de leverancierssklassen



Hoe gaan we de API voor de afstandsbediening ontwerpen…

Korte inleiding tot het Command Pattern

- In het restaurant

❶ De **KLANT** geeft aan de **SERVEERSTER** de **BESTELLING** door

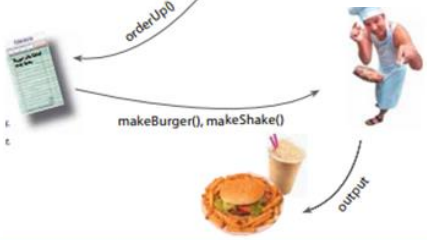
❷ De **SERVEERSTER** neemt de **BESTELLING** aan en brengt deze naar de counter en roept 'Bestelling'

❸ De **KOK** maakt de maaltijd (**BESTELLING**) klaar

De bestelling bestaat uit een bestelbon waarop de bestelling van de klant wordt geschreven

KLANT: ik wil een hamburger en een milkshake

De serveerster neemt de bestelling op en wanneer ze klaar is, roept ze de methode orderUp() aan zodat de bereiding van de maaltijd kan beginnen.



De bestelling bevat alle instructies die nodig zijn om de maaltijd te bereiden. De bestelling stuurt de kok aan via methoden als makeBurger()

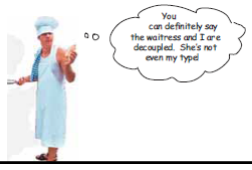
De kok volgt de instructies van de bestelling op en maakt het eten klaar.

HoGent

5

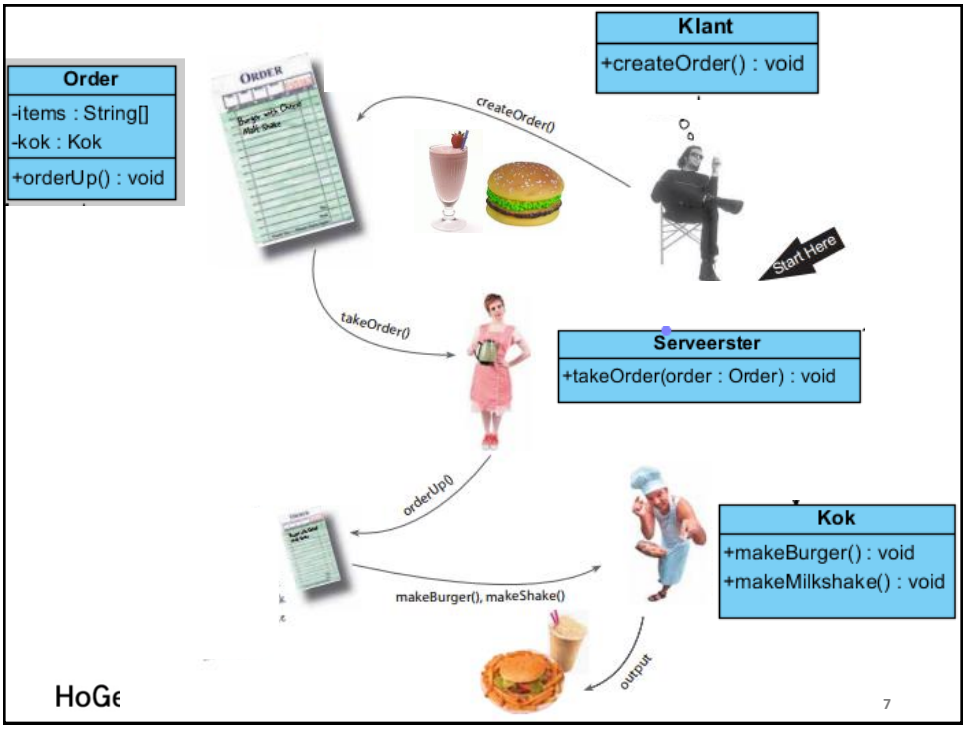
De rollen en verantwoordelijkheden

- Een bestelbon schermt een aanvraag om een maaltijd te bereiden af
  - 1 methode orderUp(). Dit schermt de acties om een maaltijd te bereiden af
  - Een referentie naar het object die de maaltijd maakt (kok)
- De taak van de serveerster is de bestelling opnemen en de methode orderUp() aanroepen, zodat de maaltijd wordt bereid
- De kok weet hoe de maaltijd bereid moet worden

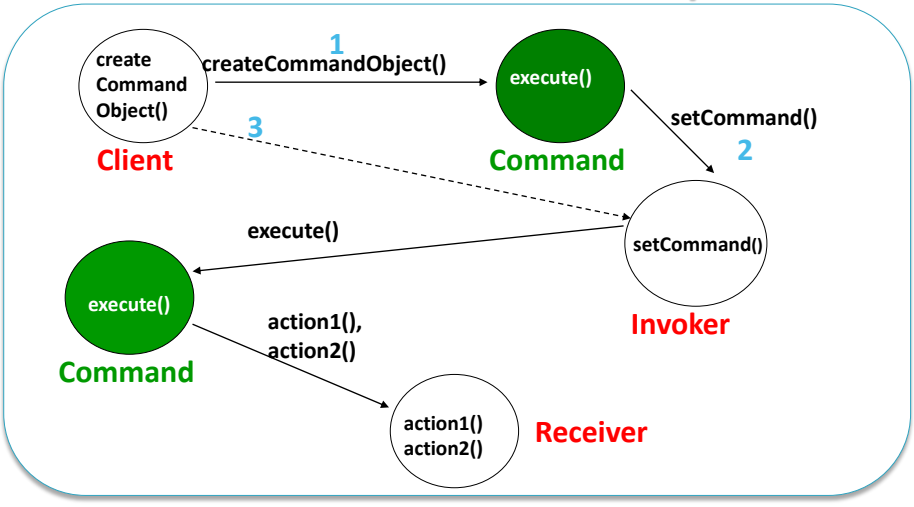


HoGent

6



Van restaurant naar Command pattern



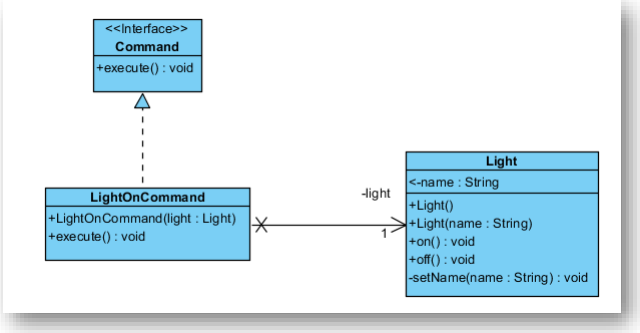
1. De Client creëert een commandobject
2. De client voert een setCommand() uit om het commandobject in de invoker op te slaan.
3. Later … de client vraagt de invoker om het command uit te voeren.

Oefening : Wie doet wat?

- Restaurant
  - Serveerster
  - Kok
  - orderUp()
  - Bestelling
  - Klant
  - takeOrder()
- Command pattern
  - Command
  - execute()
  - Client
  - Invoker
  - Receiver
  - setCommand()

Ons eerste Command object

- Implementatie van de Command interface en een Command om het licht aan te doen



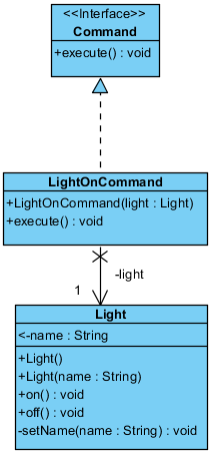
Ons eerste Command object

- Implementatie CommandInterface

```
public interface Command {
   void execute();
}
```

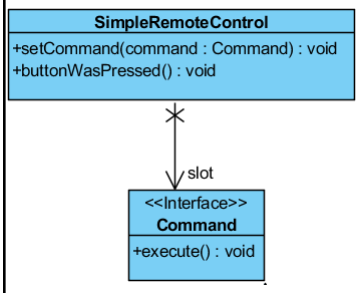
- Implementatie LightOnCommand

```
public class LightOnCommand implements Command {
   private Light light;
   public LightOnCommand(Light light) {
      this.light = light;
   }
   @Override
   public void execute() {
      light.on();
   }
}
```



Het Command object gebruiken

- we houden het eenvoudig : de afstandsbediening met maar één knop en corresponderend slot voor te besturen apparaat.




```
public class SimpleRemoteControl {

   private Command slot;

   public void setCommand(Command command) {
      slot = command;
   }

   public void buttonWasPressed() {
      slot.execute();
   }
}
```

1 slot voor onze command dat slechts 1 apparaat kan besturen.

Een methode voor het instellen van het Command dat het slot zal besturen

Deze methode wordt aangeroepen wanneer de knop wordt ingedrukt.

Werkt de afstandsbediening?

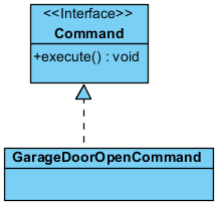
In de terminologie van het Command Pattern is dit onze Client.

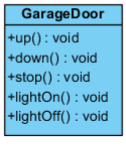
```
public class RemoteControlApp {

    public static void main(String[] args) {
        // De remote is onze Invoker.
        SimpleRemoteControl remote
                = new SimpleRemoteControl();
        // Creatie van Light-object. Dit wordt de Receiver van de aanvraag
        Light light = new Light();
        // Creatie van command en doorgave aan de Receiver (Light object).
        LightOnCommand lightOn = new LightOnCommand(light);
        //Hier wordt het Command overgedragen aan de Invoker.
        remote.setCommand(lightOn);
        //Simulatie indrukken van de knop.
        remote.buttonWasPressed();
    }
}
```

13

Oefening

- implementeer GarageDoorOpenCommand gegeven de GarageDeur






```
public class GarageDoorOpenCommand implements Command {

}
```

14

Oefening

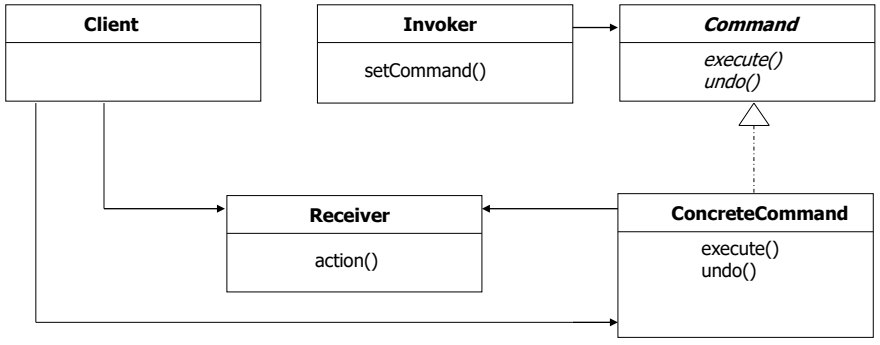
- Wat is dan de output van volgende code?

```
public class RemoteControlApp {

  public static void main(String[] args) {
        SimpleRemoteControl remote
         = new SimpleRemoteControl();
    Light light = new Light();
    LightOnCommand lightOn = new LightOnCommand(light);
    GarageDoor door = new GarageDoor();
    GarageDoorOpenCommand garageOpen =
         new GarageDoorOpenCommand(door);
    remote.setCommand(lightOn);
    remote.buttonWasPressed();
    remote.setCommand(garageOpen);
    remote.buttonWasPressed();
  }
}
```

15

Command Pattern

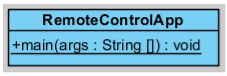
Het Command Pattern schermt een aanroep af door middel van een object, waarbij je verschillende aanroepen in verschillende objecten kun opbergen, in een queue kunt zetten of op schijf kunt bewaren; ook undo-operaties kunnen worden ondersteund.

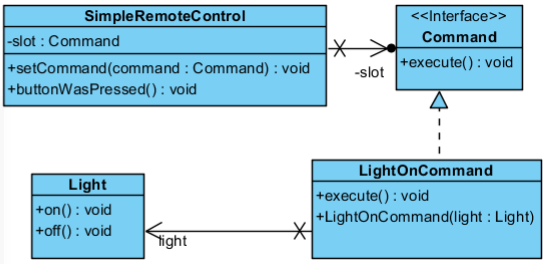


16

Command Pattern

Het Command Pattern schermt een aanroep af door middel van een object, waarbij je verschillende aanroepen in verschillende objecten kun opbergen, in een queue kunt zetten of op schijf kunt bewaren; ook undo-operaties kunnen worden ondersteund.






```
SimpleRemoteControl remote
    = new SimpleRemoteControl();
Light light = new Light();
LightOnCommand lightOn =
    new LightOnCommand(light);
remote.setCommand(lightOn);
remote.buttonWasPressed();
```

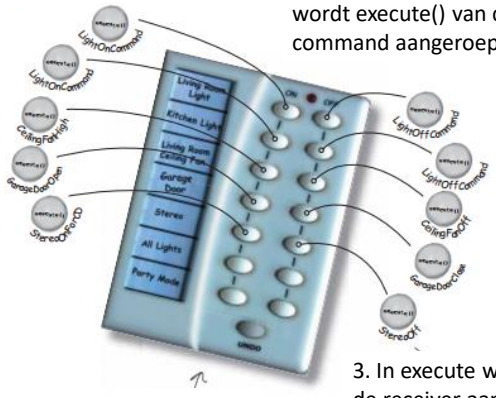
17

Commands aan slots toekennen

- Maak de volgende afstandsbediening

1. Ieder slot krijgt een command

2. Wanneer een knop wordt ingedrukt, wordt execute() van corresponderend command aangeroepen



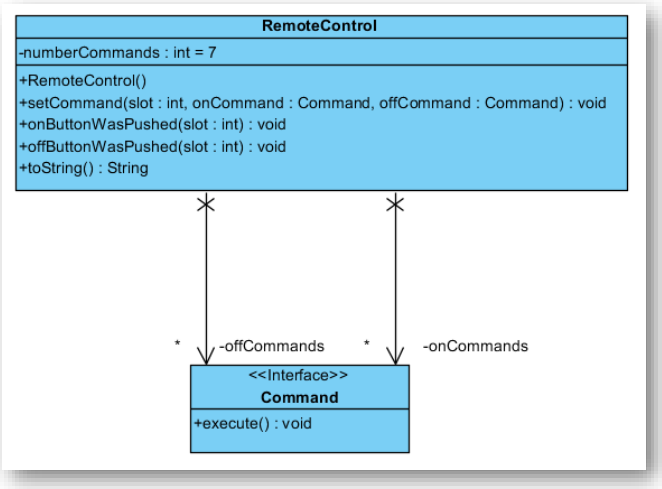
De Invoker

3. In execute worden de acties van de receiver aangeroepen

HoGe

18

Implementatie van de afstandsbediening



Implementatie van de afstandsbediening

- Wat met slots waar nog geen command aan gekoppeld is?
  - We willen niet elke keer dat we naar een slot verwijzen, in de afstandsbediening controleren of er een command geladen is

```
public void onButtonWasPushed(int slot) {
        if (onCommands[slot] != null)
                onCommands[slot].execute();
  }
```

  - Betere oplossing?

Implementatie van de afstandsbediening

- Wat met slots waar geen command aan gekoppeld is?
  - Oplossing : implementeer command dat niets doet

```
public class NoCommand implements Command {
    public void execute(){}
  }
```

  - constructor van RemoteControl : ieder slot krijgt een NoCommand-object (=> ieder slot heeft een Command)

```
Command noCommand = new NoCommand();
for (int i = 0; i < numberCommands; i++) {
   onCommands[i] = noCommand;
   offCommands[i] = noCommand;
}
```

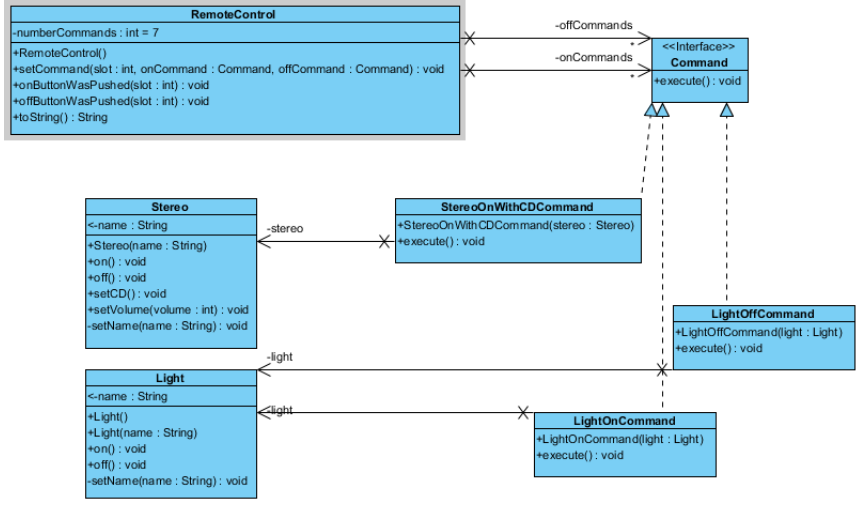

Implementatie van de afstandsbediening

```
public class RemoteControl {

//de afstandsbediening handelt zeven On- en Off-commands
   private Command[] onCommands;
   private Command[] offCommands;
   private final int numberCommands = 7;

   public RemoteControl() {
      //we instantieren en initialiseren de on-en off-arrays
      onCommands = new Command[numberCommands];
      offCommands = new Command[numberCommands];

      Command noCommand = new NoCommand();
      for (int i = 0; i < numberCommands; i++) {
         onCommands[i] = noCommand;
         offCommands[i] = noCommand;
      }
   }
```

Implementatie van de afstandsbediening

```
/*setCommand krijgt een slotpositie en een On- en een
Off-command om in dat slot op te slaan. Deze plaatst de opdrachten
voor later gebruik in de on- en off-arrays. */
  public void setCommand(int slot, Command onCommand, Command offCommand) {
    onCommands[slot] = onCommand;
    offCommands[slot] = offCommand;
  }

  public void onButtonWasPushed(int slot) {
    onCommands[slot].execute();
  }

  public void offButtonWasPushed(int slot) {
    offCommands[slot].execute();
  }
    public String toString() {...9 lines }
```


Implementatie van de commands



Implementatie van de commands

```
public class LightOffCommand implements Command {

  private Light light;

  public LightOffCommand(Light light) {
    this.light = light;
  }

  public void execute() {
    light.off();
  }
}
```

```
public class StereoOnWithCDCommand implements Command {

  private Stereo stereo;

  public StereoOnWithCDCommand(Stereo stereo) {
    this.stereo = stereo;
  }

  public void execute() {
    stereo.on();
    stereo.setCD();
    stereo.setVolume(11);
  }
}
```

HoGent

De afstandsbed. aan de praat krijgen

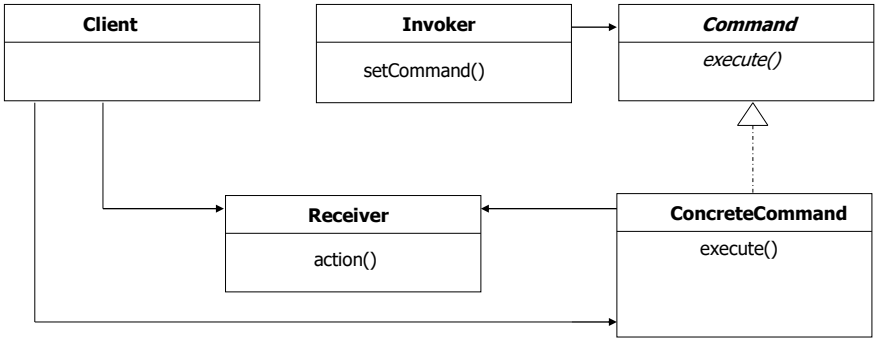
```
public static void main(String[] args) {
RemoteControl remoteControl = new RemoteControl();

Light livingRoomLight = new Light("Living Room Lighting");
Light kitchenLight = new Light("Kitchen Lighting");
Stereo stereo = new Stereo("Stereo");
//...

LightOnCommand livingRoomLightOn = new LightOnCommand(livingRoomLight);
LightOffCommand livingRoomLightOff = new LightOffCommand(livingRoomLight);
LightOnCommand kitchenLightOn = new LightOnCommand(kitchenLight);
LightOffCommand kitchenLightOff = new LightOffCommand(kitchenLight);
StereoOnWithCDCommand stereoOnWithCD
    = new StereoOnWithCDCommand(stereo);
//...
```

HoGent

De afstandsbed. aan de praat krijgen

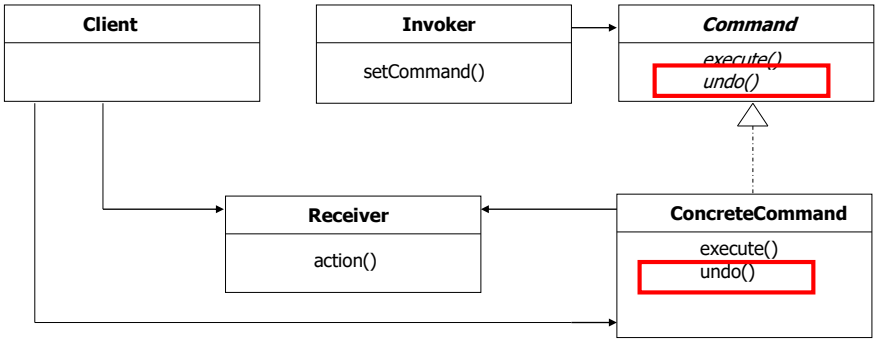
```
remoteControl.setCommand(0, livingRoomLightOn, livingRoomLightOff);
remoteControl.setCommand(1, kitchenLightOn, kitchenLightOff);
remoteControl.setCommand(2, stereoOnWithCD, new NoCommand());
//...

System.out.println(remoteControl);
for (int i = 0; i < 4; i++) {
  remoteControl.onButtonWasPushed(i);
  remoteControl.offButtonWasPushed(i);
}
```


Command Pattern



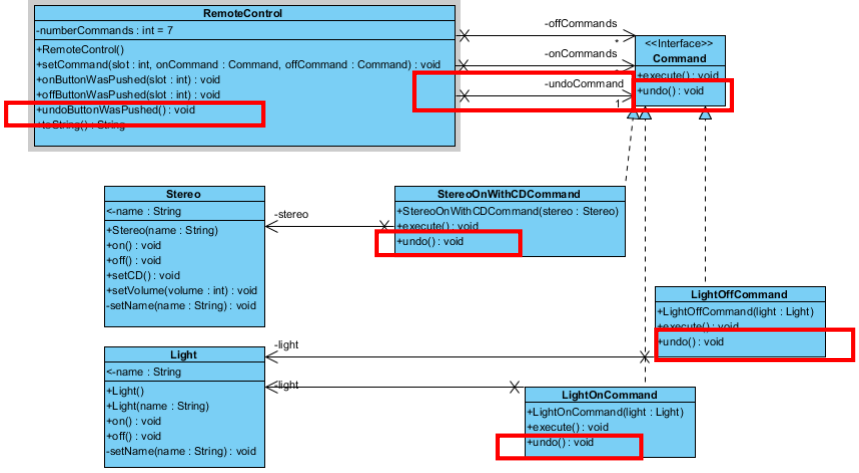
- Wat zijn we vergeten in onze implementatie?
  Pas de implementatie aan.

Command Pattern



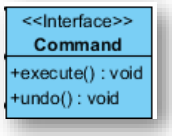
Undo

- Aanpassen ontwerp



Undo

- De Command interface




```
public interface Command {
   void execute();
   void undo();
}
```


Undo

- De concrete command klassen

```
public class LightOnCommand implements Command {

   private Light light;

   public LightOnCommand(Light light) {
      this.light = light;
   }

   @Override
   public void execute() {
      light.on();
   }
   //Execute schakelt verlichting aan, dus undo() schakelt de verlichting uit
   @Override
   public void undo() {
      light.off();
   }
}
```

Undo

- De concrete command klassen

```
public class LightOffCommand implements Command {

  private Light light;

  public LightOffCommand(Light light) {
    this.light = light;
  }
  public void execute() {
    light.off();
  }
   @Override
  public void undo() {
      ????
  }
}
```


Undo

- De concrete command klassen

```
public class NoCommand implements Command {

   @Override
   public void execute(){}

  @Override
   public void undo() {}

}
```

Undo

- De remote control

```
public class RemoteControl {
  private Command[] onCommands;
  private Command[] offCommands;
  private Command undoCommand;
  private final int numberCommands = 7;
  public RemoteControl() {
    onCommands = new Command[numberCommands];
    offCommands = new Command[numberCommands];
    Command noCommand = new NoCommand();
    for (int i = 0; i < numberCommands; i++) {
      onCommands[i] = noCommand;
      offCommands[i] = noCommand;
    }
    undoCommand = noCommand;
  }
```


Undo

- De remote control

```
public void setCommand(int slot, Command onCommand, Command offCommand) {
  onCommands[slot] = onCommand;
  offCommands[slot] = offCommand;
}
public void onButtonWasPushed(int slot) {
  onCommands[slot].execute();
  undoCommand = onCommands[slot];
}
public void offButtonWasPushed(int slot) {
  offCommands[slot].execute();
  undoCommand = offCommands[slot];
}
public void undoButtonWasPushed() {
  undoCommand.undo();
  undoCommand = new NoCommand();
}
```

Undo

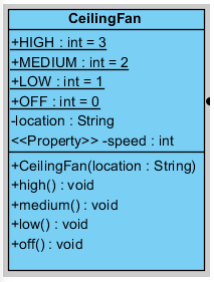
- De Undo-knop op de proef stellen

```
for (int i = 0; i < 4; i++) {
  remoteControl.onButtonWasPushed(i);
  remoteControl.offButtonWasPushed(i);
  remoteControl.undoButtonWasPushed();
```

```
Living Room Lighting is turned on
Living Room Lighting is turned off
Living Room Lighting is turned on
Kitchen Lighting is turned on
Kitchen Lighting is turned off
Kitchen Lighting is turned on
Stereo is turned on
Stereo is set for CD input
Stereo volume is set to 11
```


Toestand gebruiken voor undo

```
public class CeilingFan {

  public static final int HIGH = 3;
  public static final int MEDIUM = 2;
  public static final int LOW = 1;
  public static final int OFF = 0;
  private String location;
  private int speed;


  public CeilingFan(String location) { this.location = location; }
  public void high() { speed = HIGH; }
  public void medium() { speed = MEDIUM; }
  public void low() { speed = LOW; }
  public void off() { speed = OFF; }
  public int getSpeed() { return speed; }
}
```



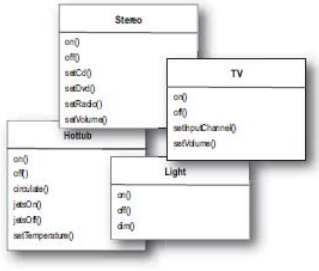
```
public class CeilingFanHighCommand implements Command {
  private CeilingFan ceilingFan;
  private int prevSpeed;

  public CeilingFanHighCommand(CeilingFan ceilingFan) {
    this.ceilingFan = ceilingFan;
  }

  @Override
  public void execute() {
    prevSpeed = ceilingFan.getSpeed();
    ceilingFan.high();
  }
@Override
public void undo() {
  if (prevSpeed == CeilingFan.HIGH) {
    ceilingFan.high();
  } else if (prevSpeed == CeilingFan.MEDIUM) {
    ceilingFan.medium();
  } else if (prevSpeed == CeilingFan.LOW) {
    ceilingFan.low();
  } else {
    ceilingFan.off();
  }
```

CeilingFanHighCommand
-prevSpeed : int -ceilingFan : CeilingFan
+CeilingFanHighCommand(ceilingFan : CeilingFan) +execute() : void +undo() : void

39

De party-modus ☺

- Wat heb je aan een afstandsbediening als je niet met één druk op de knop de lichten kan dimmen, de stereo en de TV kunt aanzetten, een DVD kunt weergeven en het bubbelbad kunt aanzetten?

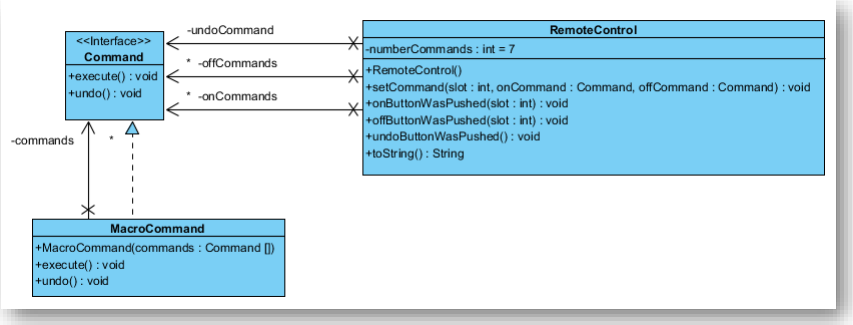


HoGent

40

Het macro-command

- Het ontwerp



Het macro-command

```
public class MacroCommand implements Command {
  private Command[] commands;
  public MacroCommand(Command[] commands) {
    this.commands = commands;
  }
  @Override
  public void execute() {
    Arrays.stream(commands).forEach(Command::execute);
  }
  /* NOTE:  these commands have to be done backwards to
   ensure proper undo functionality */
  @Override
  public void undo() {



  }
```

Het macro-command gebruiken

```
public class RemoteLoader {
    public static void main(String[] args) {

    RemoteControl remoteControl = new RemoteControl();

    Light light = new Light("Living Room");
    Tv tv = new Tv("Living Room");
    Stereo stereo = new Stereo("Living Room");
    Hottub hottub = new Hottub();

    LightOnCommand lightOn = new LightOnCommand(light);
    StereoOnCommand stereoOn = new StereoOnCommand(stereo);
    TvOnCommand tvOn = new TvOnCommand(tv);
    HottubOnCommand hottubOn = new HottubOnCommand(hottub);
    LightOffCommand lightOff = new LightOffCommand(light);
    StereoOffCommand stereoOff = new StereoOffCommand(stereo);
    TvOffCommand tvOff = new TvOffCommand(tv);
    HottubOffCommand hottubOff = new HottubOffCommand(hottub);
```

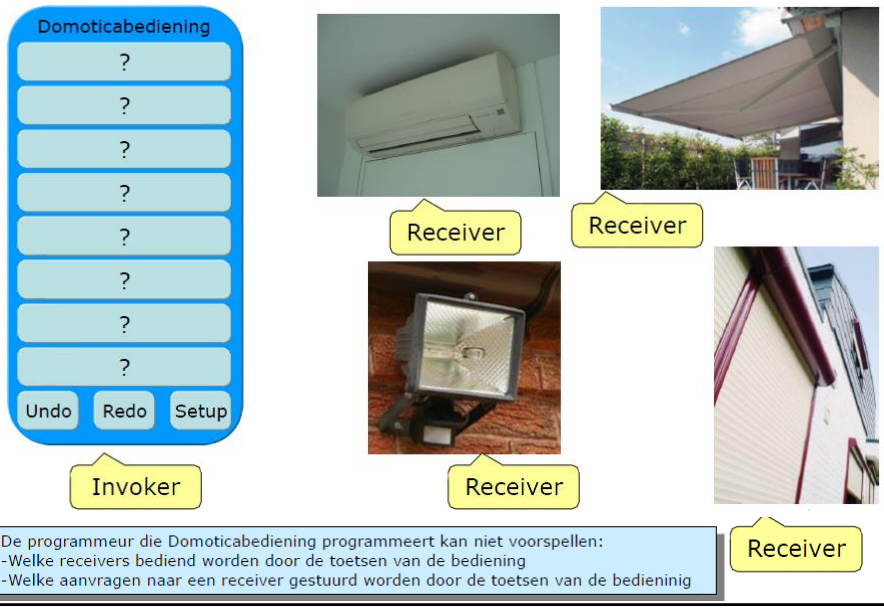

Het macro-command gebruiken

```
    Command[] partyOn = { lightOn, stereoOn, tvOn, hottubOn};
    Command[] partyOff = { lightOff, stereoOff, tvOff, hottubOff};

    MacroCommand partyOnMacro = new MacroCommand(partyOn);
    MacroCommand partyOffMacro = new MacroCommand(partyOff);

    remoteControl.setCommand(0, partyOnMacro, partyOffMacro);
    System.out.println(remoteControl);
    System.out.println("--- Pushing Macro On---");
    remoteControl.onButtonWasPushed(0);
    System.out.println("--- Pushing Macro Off---");
    remoteControl.offButtonWasPushed(0);
     }
}
```

Voorbeeld output NB-project

```
--- Pushing Macro On---
Living Room Lighting is turned on
Kitchen Room Lighting is turned on
Stereo is turned on
Stereo is set for CD input
Stereo volume is set to 11
--- Pushing Macro Off---
Living Room Lighting is turned off
Kitchen Room Lighting is turned off
Stereo is turned off
BUILD SUCCESSFUL (total time: 0 seconds)
```


Command :nut?

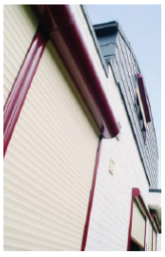


De programmeur die Domoticabediening programmeert kan niet voorspellen:
-Welke receivers bediend worden door de toetsen van de bediening
-Welke aanvragen naar een receiver gestuurd worden door de toetsen van de bedieninig

Command : nut?

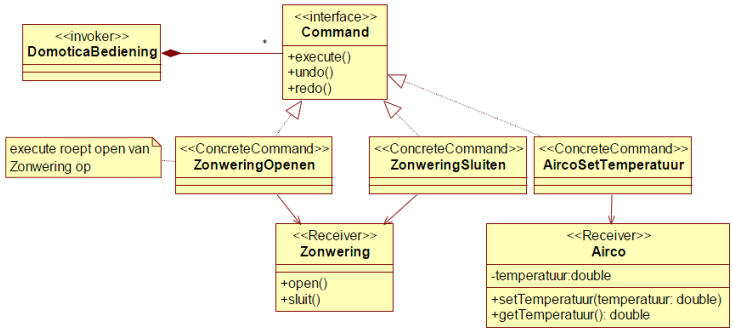
HoGent

Command nut?



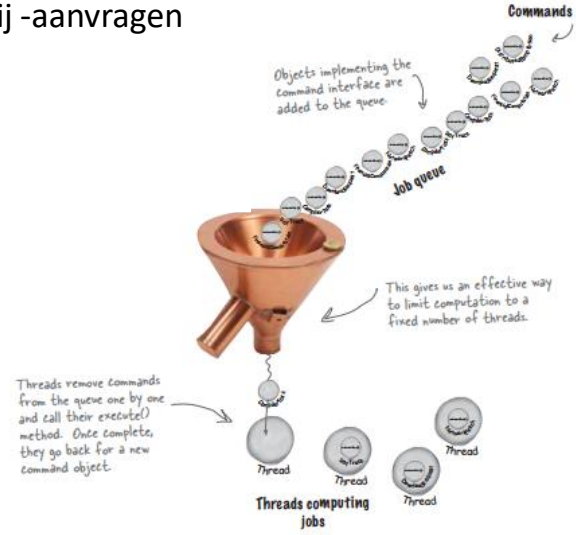
HoGent

Command : nut?



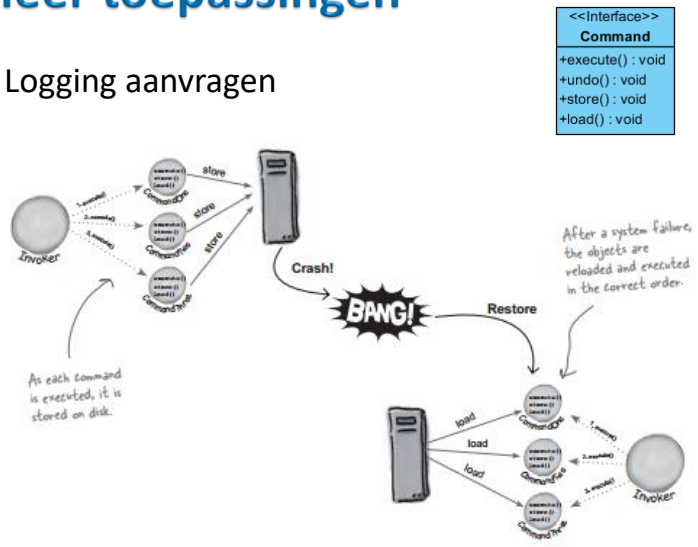
Meer toepassingen

- Wachtrij -aanvragen



Meer toepassingen

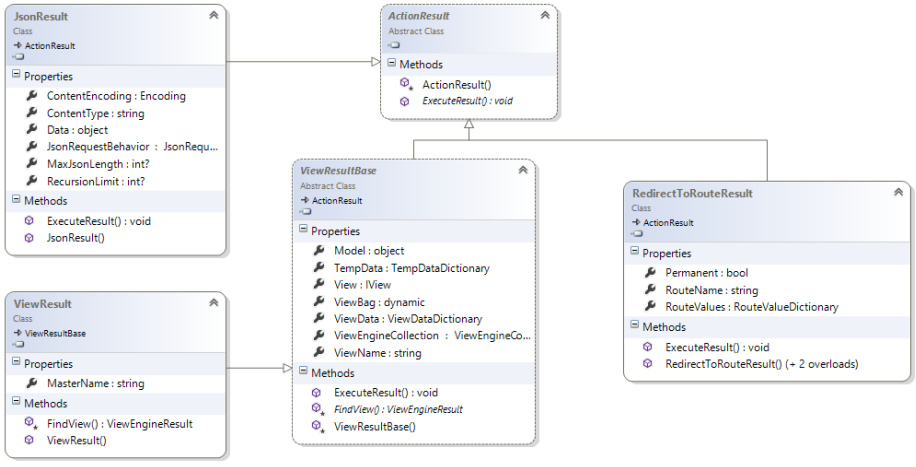
- Logging aanvragen



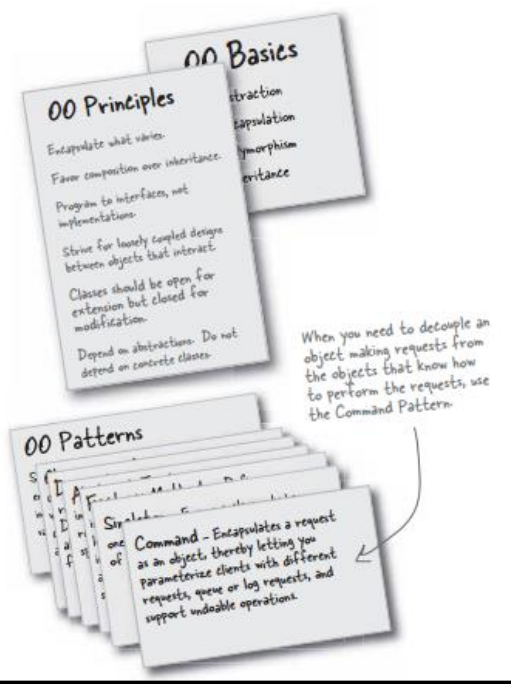
HoGent

Meer toepassingen

- ASP.NET MVC



HoGent



Command Pattern

- Een actie wordt voorgesteld als een object
- De commands zijn volledig “self-contained”
  - Ze bevatten alle info die nodig is om de actie uit te voeren
- Nieuwe commando’s kunnen eenvoudig worden toegevoegd (open/closed principe)

Referenties

- Eric, F., & Elisabeth, F. (2004). *Head First Design Patterns* (p. 629). O'Reilly Media, Inc.